

On Formal Verification in Imperative Multivalued Programming over Continuous Data Types*

Norbert Müller¹, Sewon Park², Norbert Preining³, Martin Ziegler²

¹ Trier University ² KAIST ³ JAIST

Abstract. Using fundamental ideas from [Brattka&Hertling’98] and by means of object-oriented overloading of operators, the `iRRAM` library supports imperative programming over the reals with a both sound and computable, multivalued semantics of tests. We extend Floyd-Hoare Logic to formally verify the correctness of symbolic-numerical algorithms employing such primitives for three example problems: truncated binary logarithm, 1D simple root finding, and solving systems of linear equations. This is to be generalized to other hybrid (i.e. discrete and continuous) abstract data types.

Based on mathematical logic, the Theory of Computation has devised fundamental concepts and methods that modern Software Engineering builds on. These include specification, algorithm design and analysis, models of computation, semantics of operational primitives, measures of cost, formal correctness proofs, and lower complexity bounds/intractability results — for problems over discrete structures: Common continuous realms like real numbers and functions, regularly attributed to Numerics, arguably lack behind regarding a rigorous treatment [Linz88, p.412] of its (empirically often highly successful) heuristics and ‘recipes’ [PTVF07].

Recursive Analysis provides a sound foundation of computation over real numbers, functions, and compact subsets by approximation up to guaranteed error bounds; and has been generalized to metric spaces [PERi89, Weih00]. Moreover huge strides have been made from computability theory to bit-complexity [Ko91, KORZ14, KMRZ15, KSZ14, SST16]. However the underlying Turing machine model, operating on sequences of approximations from a fixed countable dense subset, is awkward to program in practice; while the intuitive algebraic model [TuZu00] ignores rounding errors and in fact exhibits superrecursive power [BoVi99]. Reconciliating and combining the best of both worlds [MüZi14], a model of imperative computation over the reals had been suggested [BrHe98] and implemented as abstract data type `REAL` in the object-oriented programming language `C++` [Müll01]. It maintains the perspective of real numbers as entities by assigning to inequality tests a modified, namely multivalued (aka non-extensional aka fuzzy aka soft) semantics [YSS13] well-known inherent to real computation [Luck77, PaZi13].

In this work we present, and formally prove the correctness of, algorithms for three simple but natural multivalued example problems: truncated binary logarithm

$$\text{ilog}_2 : (0; \infty) \ni x \mapsto \{k \in \mathbb{Z} : 2^{k-1} < x < 2^{k+1}\} , \quad (1)$$

1D simple root finding, and nontrivially solving a given homogeneous system of linear equations $A \cdot x = 0$. Lacking tests for equality, the latter requires, in addition to continuous $A \in \mathbb{R}^{m \times n}$, the discrete input of $\text{rank}(A) \in \{0, 1, \dots, \min(n, m) - 1\}$ [ZiBr04, Zieg12, ASBZ13, KiPa16], employed in a non-trivial refinement of classical Gaussian Elimination worthwhile verifying.

Our symbolic proofs presume the underlying arithmetic primitives and multivalued tests to be correct, implemented for instance using multiple precision interval computations [FHLPZ07], and thus complement current research on the verification of floating-point arithmetic [Bold15]; cmp. [MüUh12, p.169 level 4]. The long-term goal is to extend Floyd-Hoare Logic to more general classes of hybrid algorithms, operating on both discrete and continuous abstract data types [KSZ16] such as Sobolev spaces $H^k(\Omega)$.

*First presented at CCA2016. . .

References

- [Apt81] K.R. APT: “Ten years of Hoare’s logic: A survey — Part 1”, pp.431–483 in *ACM Transactions on Programming Languages and Systems* vol.**3:4** (1981).
- [ASBZ13] K. AMBOS-SPIES, U. BRANDT, M. ZIEGLER: “Real Benefit of Promises and Advice”, pp.1–11 in *Proc. 9th Conf. on Computability in Europe (CiE’2013)*, LNCS vol.**7942**.
- [Bold15] S. BOLDO, J.-H. JOURDAN, X. LEROY, G. MELQUIOND: “Verified Compilation of Floating-Point Computations”, pp.135–163 in *Journal of Automated Reasoning* vol.**54:2** (2015).
- [BoVi99] P. BOLDI, S. VIGNA: “Equality is a Jump”, pp.49–64 in *Theoret. Comp. Sci.* vol.**219** (1999).
- [BrHe98] V. BRATTKA, P. HERTLING: “Feasible real random access machines”, pp.490–526 in *Journal of Complexity* vol.**14:4** (1998).
- [Cook78] S.A. COOK: “Soundness and Completeness of an Axiom System for Program Verification”, pp.70–90 in *SIAM J. Computing* vol.**7:1** (1978).
- [Dijk75] E.W. DIJKSTRA: “Guarded Commands, Non-Determinacy and Formal Derivation of Programs”, pp.453–457 in *Commun. ACM* vol.**18** (1975).
- [FHLPZ07] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, P. ZIMMERMANN: “MPFR: A multiple-precision binary floating-point library with correct rounding”, article 13 in *ACM Transactions on Mathematical Software* vol.**33:2** (2007).
- [KMRZ15] A. KAWAMURA, N. MÜLLER, C. RÖSNICK, M. ZIEGLER: “Computational Benefit of Smoothness: Parameterized Bit-Complexity of Numerical Operators on Analytic Functions and Gevrey’s Hierarchy”, pp.689–714 in *Journal of Complexity* vol.**31:5** (2015).
- [Ko91] K.-I. KO: *Computational Complexity of Real Functions*, Birkhäuser (1991).
- [Kohl01] U. KOHLENBACH: “On the Computational Content of the Krasnoselski and Ishikawa Fixed Point Theorems”, pp.119–145 in *Computability and Complexity in Analysis*, Springer LNCS vol.**2064** (2001).
- [KORZ14] A. KAWAMURA, H. OTA, C. RÖSNICK, M. ZIEGLER: “Computational Complexity of Smooth Differential Equations”, *Logical Methods in Computer Science* vol.**10:1** (2014).
- [KSZ14] A. KAWAMURA, F. STEINBERG, M. ZIEGLER: “Complexity of Laplace’s and Poisson’s Equation”, abstract p.231 in *Bulletin of Symbolic Logic* vol.**20:2** (2014); full version to appear in *Mathem. Structures in Computer Science*.
- [KSZ16] A. KAWAMURA, F. STEINBERG, M. ZIEGLER: “Complexity Theory of (Functions on) Compact Metric Spaces”, pp.837–846 in *Proc.31st Ann.ACM-IEEE Symp. Logic Comp.Sci. (LiCS2016)*.
- [Linz88] P. LINZ: “A Critique of Numerical Analysis”, pp.407–416 in *Bulletin American Mathematical Society* vol.**19:2** (1988).
- [Luck77] H. LUCKHARDT: “A Fundamental Effect in Computations on Real Numbers”, pp.321–324 in *Theoretical Computer Science* vol.**5** (1977).
- [Müll01] N. MÜLLER: “The iRRAM: Exact Arithmetic in C++”, pp.222–252 in *Proc. 4th Int. Workshop on Computability and Complexity in Analysis (CCA’00)*, Springer LNCS vol.**2064** (2001).
- [MüUh12] N. MÜLLER, C. UHRHAN: “Some Steps into Verification of Exact Real Arithmetic”, pp.168–173 in *Proc. 4th Int. Symp. NASA Formal Methods*, Springer LNCS vol.**7226** (2012).
- [MüZi14] N. MÜLLER, M. ZIEGLER: “From Calculus to Algorithms without Errors”, pp.718–724 in *Proc. 4th Int. Congress on Mathematical Software (ICMS2014)*, LNCS vol.**8592**.
- [PaZi13] A. PAULY, M. ZIEGLER: “Relative Computability and Uniform Continuity of Relations”, pp.1–39 in the *Journal of Logic and Analysis* vol.**5:7** (2013).
- [PERi89] M.B. POUR-EL, J.I. RICHARDS: “*Computability in Analysis and Physics*”, Springer (1989).
- [PTVF07] W.H. PRESS, S.A. TEUKOLSKY, W.T. VETTERLING, B.P. FLANNERY: *Numerical Recipes: The Art of Scientific Computing* (3rd Edition), Cambridge University Press (2007).
- [SST16] M. SCHRÖDER, F. STEINBERG, M. ZIEGLER: “Average-Case Bit-Complexity Theory of Real Functions”, pp.489–504 in *Proc. 6th Int. Conf. on Mathematical Aspects of Computer and Information Sciences (MACIS2015)*, Springer LNCS vol.**9582**.
- [TuZu00] J.V. TUCKER, J.I. ZUCKER: “Computable functions and semicomputable sets on many-sorted algebras”, pp.317–523 in *Handbook of Logic in Computer Science* vol.**5** (S. Abramsky, D.M. Gabbay, T.S.E. Maybaum Edts), Oxford Science Publications (2000).
- [Weih00] K. WEIHRAUCH: *Computable Analysis*, Springer (2000).
- [YSS13] C. YAP, M. SAGRALOFF, V. SHARMA: “Analytic Root Clustering: A Complete Algorithm Using Soft Zero Tests”, pp.434–444 in *Proc. 9th Conf. Comput. in Europe (CiE2013)*, LNCS vol.**7942**.
- [ZiBr04] M. ZIEGLER, V. BRATTKA: “Computability in Linear Algebra”, pp.187–211 in *Theoretical Computer Science* vol.**326** (2004).
- [Zieg12] M. ZIEGLER: “Real Computation with Least Discrete Advice: A Complexity Theory of Nonuniform Computability”, pp.1108–1139 in *Annals of Pure and Applied Logic* vol.**163:8** (2012).

A Appendix: Examples of Real Algorithms and Formal Verification

In Subsection A.1 we motivate and collect the computable semantics of the underlying primitives on real numbers. Subsection A.2 then presents an algorithm computing the aforementioned function ilog_2 , enriched with pre and post conditions to formally prove it totally correct. And Subsection A.3 does the same for Gaussian Elimination with a new version of pivot search that avoids (prohibitive) tests for equality and instead harnesses the input of, in addition to the real entries of the matrix A , its integer rank. Subsection A.4 considers the problem of finding the (presumed unique) root of a continuous function $f : [0; 1] \rightarrow \mathbb{R}$ with $f(0) < 0 < f(1)$. All examples begin with the classical algorithm using the naïve semantics of tests and then modify it to deal with the computable but different semantics. We conclude in Section B with some remarks concerning the in-/completeness of our generalized Floyd-Hoare calculus and future perspectives.

Remark 1 (Some Related Work).

Real-PCF and Shrad are functional (while we focus on imperative) programming languages for exact real number computation [Esca96]. Extending λ -calculus, they provide a rigorous and recursive semantics with strong theoretical foundation — waiting to be picked up by, and to compete in efficiency with, practice [BaKa08].

Reliable/interval numerics calculates enclosures to solutions of, say, partial differential equations and thus enables computer-assisted proofs to purely analytical claims [Lan82, Rump04, PWNT14] using floating-point or unbounded precision arithmetic [BLWW04]. Here, and as opposed to **iRRAM**, the user/programmer is responsible for choosing, and analyzing the gradual loss of, intermediate precision such as to guarantee the claimed output accuracy.

Validated numerics in turn aims to prove the correctness of the calculations themselves, for instance by axiomatizing floating-point arithmetic [CNR11] and by verifying their implementation [DLM09]. Note that each data type **float**, **double** etc. holds finite information; hence the verification problem is a (very large but finite, i.e.) discrete one — however with a convoluted semantics involving NaNs, underflows, overflows, exceptions etc.: cmp. IEEE 754.

Exact geometric computation considers inputs (rational or algebraic real) to be given exactly (numerator/denominator or minimal polynomial with root bounds) and thus permit tests for equality — while outputs may be approximated. This semantics violates compositionality [Yap04, p.325], though: the corner stone of modular software development. \square

A.1 Semantics of Real Operations

“*Don’t test for equality!*” may be the first lesson in Numerical Programming 101. Indeed, equality of real (and not just, say, algebraic) numbers is well-known equivalent to the Halting Problem [Weih00, EXERCISE 4.2.9]. But which comparisons are permitted, then? Strict inequality “ $x > 0$ ” for instance would allow to express equality via the Boolean combination “ $\neg(x > 0) \wedge \neg(-x > 0)$ ”: contradiction. To resolve this paradox we consider two sound semantics: partial and multivalued, both supported by **iRRAM**. The first has “ $x > 0$ ” undefined (diverge, ‘value’ \downarrow) in case $x = 0$; the second may answer arbitrarily in case $|x|$ is smaller than a given threshold. More formally recall that a partial multivalued[†] mapping $f : \subseteq X \rightrightarrows Y$ (aka search problem) is simply a relation $f \subseteq X \times Y$, considered as total function $f : X \ni x \mapsto \{y \in Y : (x, y) \in f\}$; with the understanding that an algorithm computing f may, given $x \in \text{dom}(f) = \{x : f(x) \neq \emptyset\}$, return any $y \in f(x)$. So extensionality means singleton values. We collect the semantics of (very few of) the operational primitives on real numbers, as implemented by the **iRRAM** library with data type **REAL** via object-oriented overloading:

Definition 2. $+$, $-$, $*$, $/$ denote the usual operations on real numbers: exactly, that is, without rounding errors and satisfying the axioms of a field — including distributivity. Similarly, the

[†]We prefer this term over *nondeterministic* which has evolved to a subtly different meaning; cmp. the complexity class \mathcal{NP} and [Zieg06, §5].

absolute value **abs**, (partial) algebraic and transcendental real constants and functions like **pi**, **sqr**, **exp**, **ln**, **sin**, **cos**, **tan** and their inverses coincide with their standard interpretations and in particular (as opposed to Computer Algebra Systems) exhibit all true mathematical relations, proven or not, automatically.

For $k \in \mathbb{Z}$, \succ_k and \succ denote the, respectively, multivalued and partial tests

$$(x \succ_k y) = \begin{cases} \{1\} & : x \geq y + 2^k, \\ \{0\} & : x \leq y - 2^k, \\ \{0, 1\} & \text{else} \end{cases} \quad (x \succ y) = \begin{cases} 1 & : x > y, \\ 0 & : x < y, \\ \downarrow & : x = y \end{cases}$$

The multivalued partial ‘function’ **choose** takes as arguments a finite sequence of (unevaluated[‡]) partial tests $(x_1 \succ y_1, x_2 \succ y_2, \dots, x_L \succ y_L)$, at least one promised to evaluate to true, and returns some index $1 \leq \ell \leq L$ such that $x_\ell \succ y_\ell$ holds.

The latter operation is also known as *Parallel OR* [EHS04]. In particular it holds

$$(x \succ_k y) = \mathbf{choose}(y \succ x - 2^k, x \succ y - 2^k) - 1.$$

Similarly, $\max(x, y) = (x + y)/2 + \mathbf{abs}(x - y)/2$ and $\min(x, y) = (x + y)/2 - \mathbf{abs}(x - y)/2$ can be expressed using the above operations. Note that, following the tradition/convention of **C++**, we use integers 0 and 1 to denote truth values; and thus naturally arrive at a two-sorted logic — namely over \mathbb{Z} for Booleans and counters and \mathbb{R} as continuous data type, possibly with (implicit) inclusion $\iota : \mathbb{Z} \rightarrow \mathbb{R}$ — to formulate loop invariants, pre and post conditions.

For a formal background to Floyd-Hoare Logic the reader is kindly referred to standard literature like [Apt81]. In fact, and in agreement with [Dijk75], it effortlessly supports multivalued computation; so we do not need to resort to, say, modal logic [Khan15]...

A.2 Example: Integer-Valued Binary Logarithm

We start with the trivial algorithm perusing the (uncomputable) naïve, namely both single-valued and total, semantics of tests \succ ; enriched with Hoare-style formulae for loop invariants, pre and post condition as comments:

Algorithm 1 Integer-valued logarithm using naïve test semantics

```

1: function  $\text{ilog}_2(x : \mathbb{R})$                                      // Require:  $x > 0$ 
2:    $\mathbb{Z} \ni l := 1$                                               $\{0 < x, l = 1\}$ 
3:   if  $x > 1$  then                                            $\{1 < x, l = 1\}$ 
4:      $\mathbb{R} \ni y := x$                                             $\{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}$ 
5:     while  $y > 2$  do                                          $\{2^l < y \cdot 2^{l-1} = x\}$ 
6:        $l := l + 1 ; y := y/2$                                  $\{2^{l-1} < y \cdot 2^{l-1} = x\}$ 
7:     end while                                               $\{2^{l-1} < x = y \cdot 2^{l-1} \leq 2^l\}$ 
8:   else                                                        $\{0 < x \leq 1 = 2^{l-1}\}$ 
9:     repeat
10:       $l := l - 1$                                             $\{0 < x \leq 2^l\}$ 
11:    until  $x > 2^{l-1}$                                           $\{2^{l-1} < x \leq 2^l\}$ 
12:  end if
13:  return  $l$                                                   $\{2^{l-1} < x \leq 2^l\}$ 
14: end function

```

[‡]Technically, **iRRAM** implements a dedicated data type **LAZY_BOOLEAN**, corresponding to the three-point generalized Sierpinski space $\{\downarrow, 0, 1\}$ with topology $\{\emptyset, \{0\}, \{1\}, \{0, 1\}, \{\downarrow, 0, 1\}\}$, for storing a partial test without evaluating it.

We refrain from giving the formal deductions in some proof calculus: It is easy to verify that these conditions prove partial correctness; and total correctness follows from the well-known Archimedian property of \mathbb{R} . The above algorithm thus ‘realizes’ the single-valued mapping $0 < x \mapsto \lceil \log_2(x) \rceil \in \mathbb{Z}$. However when replacing the mathematical structure \mathbb{R} with its computable version **REAL**, $>$ has to be replaced as in the following variant of the above algorithm employing the multivalued tests \succ_k for varying positive integer values of k (and thus explaining our choice of `ilog` as example as well as avoiding the temporary variable y in the second loop and deliberately using all three standard control statements, `if then else`, `while`, and `repeat until`):

Algorithm 2 Integer-valued logarithm using multivalued test semantics

<pre> 1: function <code>ilog₂</code>($x : \mathbf{REAL}$) 2: INTEGER $\ni l := 1$ 3: if $x \succ_{-1} 3/2$ then 4: $\mathbb{R} \ni y := x$ 5: while $y \succ_{-1} 5/2$ do 6: $l := l + 1 ; y := y/2$ 7: end while 8: else 9: repeat 10: $l := l - 1$ 11: until $x \succ_{l-2} 3 * 2^{l-2}$ 12: end if 13: return l 14: end function </pre>	<pre> // Require: $x > 0$ $\{0 < x, l = 1\}$ $\{1 < x, l = 1\}$ $\{1 = 2^{l-1} < y = y \cdot 2^{l-1} = x\}$ $\{2^l < y \cdot 2^{l-1} = x\}$ $\{2^{l-1} < y \cdot 2^{l-1} = x\}$ $\{2^{l-1} < x = y \cdot 2^{l-1} < 2^{l+1}\}$ $\{0 < x < 2 = 2^l\}$ $\{0 < x < 2^{l+1}\}$ $\{2^{l-1} < x < 2^{l+1}\}$ $\{2^{l-1} < x < 2^{l+1}\}$ </pre>
---	--

Recall from Definition 2 that $x \geq 2 \Rightarrow x \succ_{-1} 3/2 \Rightarrow x > 1$, $y \geq 3 \Rightarrow y \succ_{-1} 5/2 \Rightarrow y > 2$ and, for $l \leq 1$, $x \geq 4 \cdot 2^{l-2} \Rightarrow x \succ_{l-2} 3 \cdot 2^{l-2} \Rightarrow x > 2 \cdot 2^{l-2}$. Hence, again, the post conditions follow easily from the pre conditions and loop invariants.

A.3 Example: Gaussian Elimination

Consider the problem of finding a non-trivial solution $x \neq 0$ to the homogeneous system of linear equations $A \cdot x = 0$ for a given real $n \times n$ matrix A . Again we start with a classical algorithm using naïve real number tests for pivot search. Here, partial pivoting suffices to establish total correctness; yet we apply full pivoting (permuting components of x) as preparation for the below variant employing **REAL** numbers and tests. In the algebraic model, any non-zero element can equally serve as pivot; again anticipating the later refined model, the following subroutine searches for (the first) one of maximum absolute value:

Algorithm 3 Full pivot search using naïve test semantics

```

1: procedure CHOOSEPIVOT( $n : \mathbb{N}$ ,  $k : \mathbb{N}$ ,  $B[n, n] : \mathbb{R}$ , var  $pi : \mathbb{N}$ , var  $pj : \mathbb{N}$ )
2: // Return index  $(pi, pj)$  of first element of max. absolute value in sub-matrix  $B' := B[k \dots n, k \dots n]$ 
3:   var  $i, j : \mathbb{N}$ ; var  $t : \mathbb{R}$ 
4:    $pi := k$ ;  $pj := k$ ;  $t := \text{abs}(B[k, k])$ 
5:   for  $i := k$  to  $n$  do
6:     for  $j := k$  to  $n$  do
7:       if  $\text{abs}(B[i, j]) > t$  then  $t := \text{abs}(B[i, j])$ ;  $pi := i$ ;  $pj := j$  end if
8:     end for
9:   end for
10: end procedure                                     //  $\text{abs}(B[pi, pj]) = \max \{ \text{abs}(B[i, j]) : k \leq i, j \leq n \}$ 

```

Like in Pascal we use the keyword **var** to indicate arguments passed by reference rather than by value. Let us now proceed to the main elimination algorithm employing the above subroutine. In the sequel abbreviate $\pi(x) := (x_{\pi(1)}, \dots, x_{\pi(n)})$ for $x \in \mathbb{R}^n$ and $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$. We deliberately avoid optimizations (such as doing calculations in place) in order to simplify the invariants for verification. Integer quantification is to be understood over $\{1, \dots, n\}$.

Algorithm 4 Gaussian Elimination using naïve test semantics

```

1: function GAUSS( $n : \mathbb{N}$ ,  $A[n, n] : \mathbb{R}$ , var  $x[n] : \mathbb{R}$ )                                     // Require:  $\text{rank}(A) < n$ 
2:   var  $i, j, k, pi, pj, \pi[n] : \mathbb{N}$ ; var  $t, B[n, n] : \mathbb{R}$                                      // Return some  $x \neq 0$  such that  $A \cdot x = 0$ 
3:   for  $i := 1$  to  $n$  do                                                                 // Initialization  $B := A$ ,  $\pi := \text{id}$ 
4:      $\pi[i] := i$ ; for  $j := 1$  to  $n$  do  $B[i, j] := A[i, j]$  end for
5:   end for
6:    $k := 0$                                                                  // Convert  $B$  to reduced row echelon form:
7:   repeat    $\{ \forall y \in \mathbb{R}^n : A \cdot y = 0 \Leftrightarrow B \cdot \pi(y) = 0, \quad \forall j \leq k : B[j, j] = 1 \wedge \forall i > j : B[i, j] = 0 \}$ 
8:      $k := k + 1$ 
9:     CHOOSEPIVOT( $n, k, B, pi, pj$ ) // Find  $pi, pj$  s.t.  $\text{abs}(B[pi, pj]) = \max \{ \text{abs}(B[i, j]) : k \leq i, j \leq n \}$ 
10:    if  $B[pi, pj] \neq 0$  then
11:      for  $j := 1$  to  $n$  do  $\text{swap}(B[k, j], B[pi, j])$  end for // Exchange rows  $\#k$  and  $\#pi$ .
12:      for  $i := 1$  to  $n$  do  $\text{swap}(B[i, k], B[i, pj])$  end for // Exchange columns  $\#k$  and  $\#pj$ .
13:       $\text{swap}(\pi[k], \pi[pj])$ 
14:      for  $j := k$  to  $n$  do // Scale row  $\#k$  by  $1/B[k, k]$ 
15:         $B[k, j] := B[k, j] / B[k, k]$  // and subtract  $B[i, k]$ -fold from rows  $\#i = k + 1 \dots n$ :
16:        for  $i := k + 1$  to  $n$  do  $B[i, j] := B[i, j] - B[i, k] * B[k, j]$  end for
17:      end for
18:    end if
19:    until  $k = n \vee B[k, k] = 0$  //  $\{k = n \vee k - 1 = \text{rank}(A) = \text{rank}(B)\}$ 
20:    if  $B[k, k] \neq 0$  then exit end if // error: matrix is regular
21:     $x[\pi[n]] := 1$  // Back-substitution for  $x$ , taking into account permutation  $\pi$ :
22:    for  $i := n - 1$  downto  $1$  do
23:       $t := 0$ ; for  $j := n$  downto  $i + 1$  do  $t := t + B[i, j] * x[\pi[j]]$  end for
24:       $x[\pi[i]] := -t$  //  $\{ \forall j \geq i : (B \cdot \pi(x))_j = 0 \}$ 
25:    end for
26:    return  $k - 1$  //  $\text{rank}(A) = \text{rank}(B) = k - 1$ 
27: end function

```

We refrain from giving a formal derivation of correctness of Algorithm 4 but recall that the loop invariant of line 7 is maintained due to the following

Fact 3 Fix $A \in \mathbb{R}^{n \times n}$.

- a) Possibly by definition, $\text{rank}(A)$ coincides with $n - \dim \text{kernel}(A)$.
- b) Permuting A 's rows does not affect its kernel.
- c) Non-trivially scaling a row of A does not affect its kernel, either.
- d) Adding a scalar multiple of one row to another does not affect the kernel.
- e) Permuting A 's columns permutes the dimensions of $\text{kernel}(A)$ accordingly.
- f) Before/after the k -th iteration of the loop comprising lines 7 to 19, B has the following form:

$$k \text{ rows } \left\{ \begin{pmatrix} 1 & * & * & \cdots & * & \cdots & * \\ 0 & 1 & * & * & \cdots & * & \cdots & * \\ 0 & 0 & 1 & * & \cdots & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & * & \cdots & * \\ 0 & 0 & \cdots & \cdots & 1 & * & \cdots & * \\ 0 & 0 & \cdots & \cdots & 0 & * & \cdots & * \\ \vdots & \vdots & \cdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & * & \cdots & * \end{pmatrix} \right\} =: B' \quad (2)$$

Due to the surprisingly few differences (underlined), correctness similarly follows for the real variant below. Lacking tests for in/equality, finding some $x \neq 0$ with $A \cdot x = 0$ is ('discontinuous' and hence) not computable from given $A \in \mathbb{R}^{2 \times 2}$ of promised $\text{rank}(A) < 2$; but the problem does become computable when given, in addition to $A \in \mathbb{R}^{n \times n}$, the integer $\text{rank}(A) \in \{0, \dots, n-1\}$ [ZiBr04] — and this enrichment is optimal [Zieg12]: As opposed to the naïve case, the rank cannot be computed nor verified but must be part of the input and relied on! The below algorithm realizes such a computation using the primitives from Definition 2. Its use of full pivoting is confirmed by numerical experience [TrSc90].

Algorithm 5 Gaussian Elimination using multivalued test semantics

```

1: procedure GAUSS( $n, r$  : INTEGER,  $A[n, n]$  : REAL, var  $x[n]$  : REAL)           // Require:  $\text{rank}(A) = r$ 
2:   var  $i, j, k, pi, pj, \pi[n]$  :  $\mathbb{N}$ ; var  $t, B[n, n]$  :  $\mathbb{R}$                      // Return some  $x \neq 0$  such that  $A \cdot x = 0$ 
3:   for  $i := 1$  to  $n$  do                                                     // Initialization  $B := A, \pi := \text{id}$ 
4:      $\pi[i] := i$ ; for  $j := 1$  to  $n$  do  $B[i, j] := A[i, j]$  end for
5:   end for
6:   for  $k := 1$  to  $r$  do                                                     // Convert  $B$  into reduced row echelon form:
7:     CHOOSEPIVOT( $n, k, B, pi, pj$ )                                           // Find  $pi, pj$  such that  $B[pi, pj] \neq 0$ .
8:     for  $j := 1$  to  $n$  do  $\text{swap}(B[k, j], B[pi, j])$  end for                 // Exchange rows  $\#k$  and  $\#pi$ .
9:     for  $i := 1$  to  $n$  do  $\text{swap}(B[i, k], B[i, pj])$  end for                 // Exchange columns  $\#k$  and  $\#pj$ .
10:     $\text{swap}(\pi[k], \pi[pj])$ 
11:    for  $j := k$  to  $n$  do                                                     // Scale row  $\#k$  by  $1/B[k, k]$ 
12:       $B[k, j] := B[k, j]/B[k, k]$                                            // and subtract  $B[i, k]$ -fold from rows  $\#i = k+1 \dots n$ :
13:      for  $i := k+1$  to  $n$  do  $B[i, j] := B[i, j] - B[i, k] * B[k, j]$  end for
14:    end for
15:  end for
16:   $x[\pi[n]] := 1$                                                            // Back-substitution for  $x$ , taking into account permutation  $\pi$ :
17:  for  $i := n-1$  downto  $1$  do
18:     $t := 0$ ; for  $j := n$  downto  $i+1$  do  $t := t + B[i, j] * x[\pi[j]]$  end for
19:     $x[\pi[i]] := -t$ 
20:  end for
21: end procedure

```

Fact 3f) asserts the sub-matrix $B' := B[k \dots n, k \dots n]$ to be not identically zero for $1 \leq k \leq r$ as required in line 7 when calling the modified pivot search Algorithm 6 below. As opposed to the naïve case, here one (provably) cannot in general find the index of an element attaining the maximum, although the maximum itself is trivially computable. Instead we suffice with some element strictly exceeding half of the maximum: which exists if the (sub) matrix is non-zero. Observe the multivalued choice in line 8:

Algorithm 6 Full pivot search using multivalued test semantics

```

1: procedure CHOOSEPIVOT( $n, k : \text{INTEGER}, B[n, n] : \text{REAL}, \text{var } pi, pj : \text{INTEGER}$ )
2:   var  $i, j : \text{INTEGER}$  ; var  $s, t : \text{REAL}$  ;  $t := 0$     // Require:  $B' := B[k \dots n, k \dots n]$  not all zero
3:   for  $i := k$  to  $n$  do                                // Calculate maximum absolute value of square sub-matrix  $B'$ :
4:     for  $j := k$  to  $n$  do  $t := \max(t, \text{abs}(B[i, j]))$  end for
5:   end for                                // Now find index of some element whose absolute value exceeds half of  $t$ :
6:   for  $i := k$  to  $n$  do
7:     for  $j := k$  to  $n$  do
8:        $s := \text{abs}(B[i, j])$  ; if  $\text{choose}(s > t/2, t > s) = 1$  then  $pi := i$  ;  $pj := j$  end if
9:     end for
10:  end for    // Return index of some element of at least half the maximum absolute value in  $B'$ .
11: end procedure

```

The first phase calculates $t = \max \{ \text{abs}(B[i, j]) : k \leq i, j \leq n \}$ due to associativity $\max\{a, b, c\} = \max\{a, \max\{b, c\}\}$. By hypothesis it holds $t > 0$; hence every $s \in \mathbb{R}$ satisfies $s > t/2$ or $s > t$ (or both). Therefore line 8 is indeed total according to the semantics of **choose** from Definition 2! Moreover, for i and j with $s = \text{abs}(B[i, j]) = t$, $\text{choose}(s > t/2, t > s)$ is guaranteed to return 1. Variables pi, pj thus do get assigned at least once to return the index of a non-zero pivot: total correctness of Algorithm 6.

A.4 Example: Simple Root Finding

According to the Intermediate Value Theorem, any (computable and thus) continuous function $f : [0; 1] \rightarrow [-1; 1]$ with $f(0) < 0 < f(1)$ has at least one root $x \in (0; 1)$ with $f(x) = 0$. Consider the problem of approximating *some* such root, that is, given f as black box as well as $n \in \mathbb{Z}$, producing some $c \in [0; 1]$ such that $|x - c| \leq 2^{-n}$. See [SaMe16] for the state of the art when f is restricted to polynomials. Without bounding f 's modulus of continuity, this is not to be confused with the problem of finding an *approximate root*, that is, some c such that $|f(c)| \leq 2^{-n}$. The following algorithm uses bisection to solve the former problem with verification:

Algorithm 7 Bisection for approximating a root of a given continuous function with sign change

```

1: function FINDROOT( $n : \mathbb{Z}, f : \mathbb{R} \rightarrow \mathbb{R}$ )                // Require:  $f$  continuous,  $f(0) \leq 0 \leq f(1)$ .
2:   var  $k : \mathbb{Z}$  ; var  $a, b, c, u, v, w : \mathbb{R}$ 
3:    $a := 0$  ;  $b := 1$  ;  $u := f(a)$  ;  $v := f(b)$  ;  $k := 0$ 
4:   if  $\neg(0 > u \wedge v > 0)$  then exit end if
5:   while  $b - a > 2^{-n}$  do                                 $\{0 \leq a < b \leq 1, b - a \leq 2^{-k}, u = f(a) \leq 0 \leq f(b) = v\}$ 
6:      $k := k - 1$  ;  $c := (a + b)/2$  ;  $w := f(c)$ 
7:     if  $w > 0$  then  $b := c$  ;  $v := w$  else  $a := c$  ;  $u := w$  end if
8:   end while
9:   return  $c$                                                $\{\exists x \in [0; 1] : f(x) = 0 \wedge |x - c| \leq 2^{-n}\}$ 
10: end function

```

Its total correctness follows from the aforementioned Intermediate Value Theorem. In case c in line 6 already is a root, the test in line 7 will be undefined with respect to the computable semantics according to Definition 2. The following algorithm thus employs *trisection* — under the hypothesis that the root is unique. We refrain here from generalizing to f whose roots are all simple, and otherwise refer to a famous counterexample due to Ernst Specker [Weih00, THEOREM 6.3.8.2]...

Algorithm 8 Trisection for approximating unique root of given cont.function with sign change

```

1: function FINDROOT( $n$  : INTEGER,  $f$  : REAL  $\rightarrow$  REAL)      // Require:  $f$  continuous has unique root,
2:   var  $k$  : INTEGER ; var  $a, b, c, d, u, v, w, y$  : REAL      // Require:  $f(0) < 0 < f(1)$ .
3:    $a := 0$  ;  $b := 1$  ;  $u := f(a)$  ;  $v := f(b)$  ;  $k := 0$ 
4:   while  $b - a >_{n-2} 3 * 2^{n-2}$  do            $\{0 \leq a < b \leq 1, b - a = (3/2)^k, u = f(a) < 0 < f(b) = v\}$ 
5:      $k := k - 1$  ;  $c := b/3 + 2 * a/3$  ;  $d := 2 * b/3 + a/3$  ;  $w := f(c)$  ;  $y := f(d)$ 
6:     if  $\text{choose}(0 > u * y, 0 > w * v) = 1$  then  $b := d$  ;  $v := y$  else  $a := c$  ;  $u := w$  end if
7:   end while
8:   return  $c$                                             $\{\exists x \in [0; 1] : f(x) = 0 \wedge |x - c| \leq 2^n\}$ 
9: end function

```

B Conclusion

In the above examples it is easy to manually verify post conditions and loop invariants. Since they also lack recursive calls, we have refrained from specifying a formal proof calculus/deductive system for automatic verification. In fact it is well-known that, due to Gödel’s Incompleteness Theorem, already over integers alone no sound recursively enumerable such system can be complete in the sense of asserting every true Hoare Triple $\{P\}A\{Q\}$ to be derivable [Cook78, §6]. So the best to hope for is ‘semantic completeness’ in the sense that, for any pre condition P and algorithm A , the so-called *strongest post condition* Q at least be expressible.

Remark 4. a) All loop invariants, pre and post conditions employed in the above examples are formulated in, apart from integer arithmetic, the language of real closed fields. Indeed, the graph $\{(x, \text{abs}(x)) : x\}$ of the absolute value function (and by the remark following Definition 2 also max and min) can be expressed in that language, for instance as $\{(x, y) : y \geq 0 \wedge (y = x \vee y = -x)\}$. And, as opposed to that of integers, this first-order theory *is* (like that of Booleans) complete.

- b) On the other hand, with integer matrix dimension n as parameter, the number of real variables $B[i, j]$ quantified over in Subsection A.3 is not constant and hence constitute (though rather special) instances of infinitary logic; cmp. [Cuck92, DEFINITIONS 2.8+3.3].
- c) Regarding n as parameter, the computational cost of verifying given loop invariants and pre/post conditions of a given real algorithm over arithmetic operations (and possibly **abs** and **sqr**t) seems to correspond to the complexity class $\mathcal{BP}(\text{NP}_{\mathbb{R}}^0)$ between NP and PSPACE, well-known from other contexts; cmp. [MeMi97, DEFINITION 3.2] and [HeZi11, HSZ13].
- d) Adding the exponential function as primitive renders real verification related to the (open) Tarski exponential function problem.

To see one direction of d), consider the following fragment:

Algorithm 9 Hardness of the real verification problem

```

1: function HARD2VERIFY( $n$  : INTEGER,  $x[n]$  : REAL)
2:   var  $y$  : REAL           // Require:  $q(x) = 0$  for some fixed  $n$ -variate integer polynomial  $q$ 
3:    $y := p(x)$               // Evaluate some fixed  $n$ -variate integer polynomial  $p$ 
4:   if  $y > 0$  then return 1 else return -1 end if           // Return  $\text{sign}(y) \neq 0$ 
5: end function

```

According to the semantics of real test “ $>$ ” from Definition 2, this algorithm is total iff $q(x) = 0 \Rightarrow p(x) \neq 0$ holds: for given n and q, p a well-known $\text{coBP}(\text{NP}_{\mathbb{R}}^0)$ -complete problem [BCSS98, §5.4].

Further References

- [BaKa08] A. BAUER, I. KAVKLER: “Implementing Real Numbers With RZ”, pp.365–384 in *Proc. 4th Int. Conf. Computability and Complexity in Analysis* (CCA 2007), ENTCS vol.**202** (2008).
- [BCSS98] L. BLUM, F. CUCKER, M. SHUB, S. SMALE: “*Complexity and Real Computation*”, Springer (1998).
- [BLWW04] F. BORNEMANN, D. LAURIE, S. WAGON, J. WALDVOGEL: “The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing”, SIAM (2004); <http://www.siam.org/books/100digitchallenge/>
- [CNR11] P. COLLINS, M. NIQUI, N. REVOL: “A Validated Real Function Calculus”, pp.437–467 in *Mathematics in Computer Science* vol.**5** (2011).
- [Cuck92] F. CUCKER: “The Arithmetical Hierarchy over the Reals”, pp.375–395 in *Journal of Logic and Computation* vol.**2:3** (1992).
- [DLM09] M. DAUMAS, D. LESTER, C. MUOZ: “Verified Real Number Calculations: A Library for Interval Arithmetic”, pp.226–237 in *IEEE Transactions on Computers* vol.**58:2** (2009).
- [EHS04] M. ESCARDÓ, M. HOFMANN, T. STREICHER: “On the non-sequential nature of the interval-domain model of real-number computation”, pp.803–814 in *Mathematical Structures in Computer Science* vol.**14:6** (2004).
- [Esca96] M.H. ESCARDO: “PCF extended with real numbers”, pp.79–115 in *Theoretical Computer Science* vol.**162:1** (1996).
- [Farj06] A. FARJUDIAN: “Shrad: A Language for Sequential Real Number Computation”, pp.49–105 in *Theory of Computing Systems*, vol.**41:1** (2007).
- [HeZi11] C. HERRMANN, M. ZIEGLER: “Computational Complexity of Quantum Satisfiability”, pp.175–184 in *Proc. 26th Annual IEEE Symposium on Logic in Computer Science* (2011).
- [HSZ13] C. HERRMANN, J. SOKOLI, M. ZIEGLER: “Satisfiability of cross product terms is complete for real nondeterministic polytime Blum-Shub-Smale machines”, in *Proc. 6th Int. Conf. Machines, Computations and Universality*, pp.85–92 of *Electronic Proceedings in Theoretical Computer Science* vol.**128** (2013).
- [Khan15] M.A. KHAN: “A Modal Logic for Non-deterministic Information Systems”, pp.119–131 in *Proc. 6th Indian Conference on Logic and Its Applications* (ICLA2015), LNCS vol.**8923**.
- [KiPa16] T. KIHARA, A. PAULY: “Dividing by zero – how bad is it, really?”, *Proc. 41st Int. Symp. on Mathematical Foundations of Computer Science* (MFCS2016); vol.**58** in *LIPICs: Leibniz International Proceedings in Informatics* (2016).
- [KoVo04] M. KOROVINA, A. VOROBJOV: “Pfaffian Hybrid Systems”, pp.430–441 in *Proc. 18th International Workshop on Computer Science Logic and 13th Annual Conference of the EACSL*, Springer LNCS vol.**3210** (2004).
- [Lanf82] O.E. LANFORD III: “A Computer-Assisted Proof of the Feigenbaum Conjectures”, pp.427–434 in *Bull. Amer. Math. Soc.* vol.**6:3** (1982).
- [MeMi97] K. MEER, C. MICHAUX: “A Survey on Real Structural Complexity Theory”, pp.113–148 in *Bulletin of the Belgian Mathematical Society* vol.**4** (1997).
- [PWNT14] M. PLUM, Y. WATANABE, K. NAGATOU, M.T. NAKAO: “Verified Computations of Eigenvalue Enclosures for Eigenvalue Problems in Hilbert Spaces”, pp.975–992 in *SIAM Journal of Numerical Analysis* vol.**52:2** (2014).
- [Rump04] S.M. RUMP: “Computer-Assisted Proofs I”, pp.2–11 in *Bulletin of the Japan Society for Industrial and Applied Mathematics* vol.**14:3** (2004).

- [SaMe16] M. SAGRALOFF, K. MEHLHORN: “Computing real roots of real polynomials”, pp.46–86 in *Journal of Symbolic Computation* vol.**73** (2016).
- [TrSc90] L.N. TREFETHEN, R.S. SCHREIBER: “Average-case stability of Gaussian elimination”, pp. 335–360 in *SIAM Journal on Matrix Analysis and Applications* vol.**11:3** (1990).
- [Yap04] C.-K. YAP: “On Guaranteed Accuracy Computation”, pp.322–373 in *Geometric Computation* (Falai Chen and Dongming Wang Edts), World Scientific Publishing (2004).
- [Zieg06] M. ZIEGLER: “Revising Type-2 Computation and Degrees of Discontinuity”, pp.255–274 in *Proc. 3rd International Conference on Computability and Complexity in Analysis* (CCA’06), Electronic Notes in Theoretical Computer Science vol.**167** (2007).